# EECS 2011 M: Fundamentals of Data Structures

**Suprakash Datta**
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/2011M`
Also on Moodle

# Iterators and the Java Collections Framework

Relevant Sections

- Iterators (Ch. 7.4)
- The Java Collections Framework (Ch. 7.5)

Note: Some slides in this presentation have been adapted from the authors' slides.

# Ch 7.4: Iterators in Java

- enables us to traverse through a collection and to remove elements from the collection selectively
- created by calling the collection's iterator method. Suppose collection is an instance of a `Collection`. Then to print out each element on a separate line:

```
Iterator<E> it = collection.iterator();
while (it.hasNext())
    System.out.println(it.next());
```

- Note that next() does two things:
  - Returns the current element (initially the first element)
  - Steps to the next element and makes it the current element.

# Iterable Interface

- Java defines a parameterized interface, named Iterable, that includes the following single method:
  - `iterator()`: Returns an iterator of the elements in the collection.
- An instance of a typical collection class in Java, such as an ArrayList, is iterable (but not itself an iterator); it produces an iterator for its collection as the return value of the `iterator()` method.
- Each call to `iterator()` returns a new iterator instance, and allows multiple (even simultaneous) traversals of a collection.

# Iterator Interface

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove(); //optional
}
```

- `hasNext()`: returns true if there are more elements
- `next()`: returns the next element; throws exception if there are no more
- `remove()`: removes the last element that was returned by next
  - remove may be called only once per call to next otherwise throws an exception
  - Iterator.remove is the only safe way to modify a collection during iteration

# for-each Loop

Java's Iterable class enables the "for-each" loop syntax:

```
for (Xtype o : collection)
    System.out.println(o);
}
```

- prints each element of the collection on a new line.
- This code is just shorthand: it compiles to use `o.iterator()`

```
Iterator <Xtype> iter = collection.iterator();
while(iter.hasNext()){
    Xtype  x = iter.next();
    System.out.println(x);
}
```

# More on Iterators

- Could represent a sequence, set or map
- Could be implemented using arrays or linked lists.

`ListIterator` Extends `Iterator`

- access to the integer position (index) of elements
- forward and backward traversal
- modification and insertion of elements
- supports the following methods: `add(e)`, `hasNext()`, `hasPrevious()`, `previous()`, `next()`, `nextIndex()`, `previousIndex()`, `set(e)`, `remove()`

# Ch 7.5: The Java Collections Framework

- good example of how to apply the principles of object-oriented software engineering to the design of classical data structures.
- A coupled set of classes and interfaces that implement commonly reusable collection data structures
- Collection: An object (or container) that groups multiple elements into a single unit.

# Collections Framework

A unified architecture for representing and manipulating collections. Includes:

- Interfaces: A hierarchy of ADTs.
- Implementations
- Algorithms: on objects that implement collection interfaces; these are polymorphic (the same method can be used on many different implementations of the appropriate collection interface)

# Java Collections Framework

In: Package java.util
More details in

- Javadoc, provided with your java distribution.
- Comments and code in the specific java.util.*.java files
- The Collections Java tutorial, available at http://docs.oracle.com/javase/tutorial/collections/inde
- Chan et al, The Java Class Libraries, Second Edition

# Java Collections Framework - 2

In: Package java.util
More details in

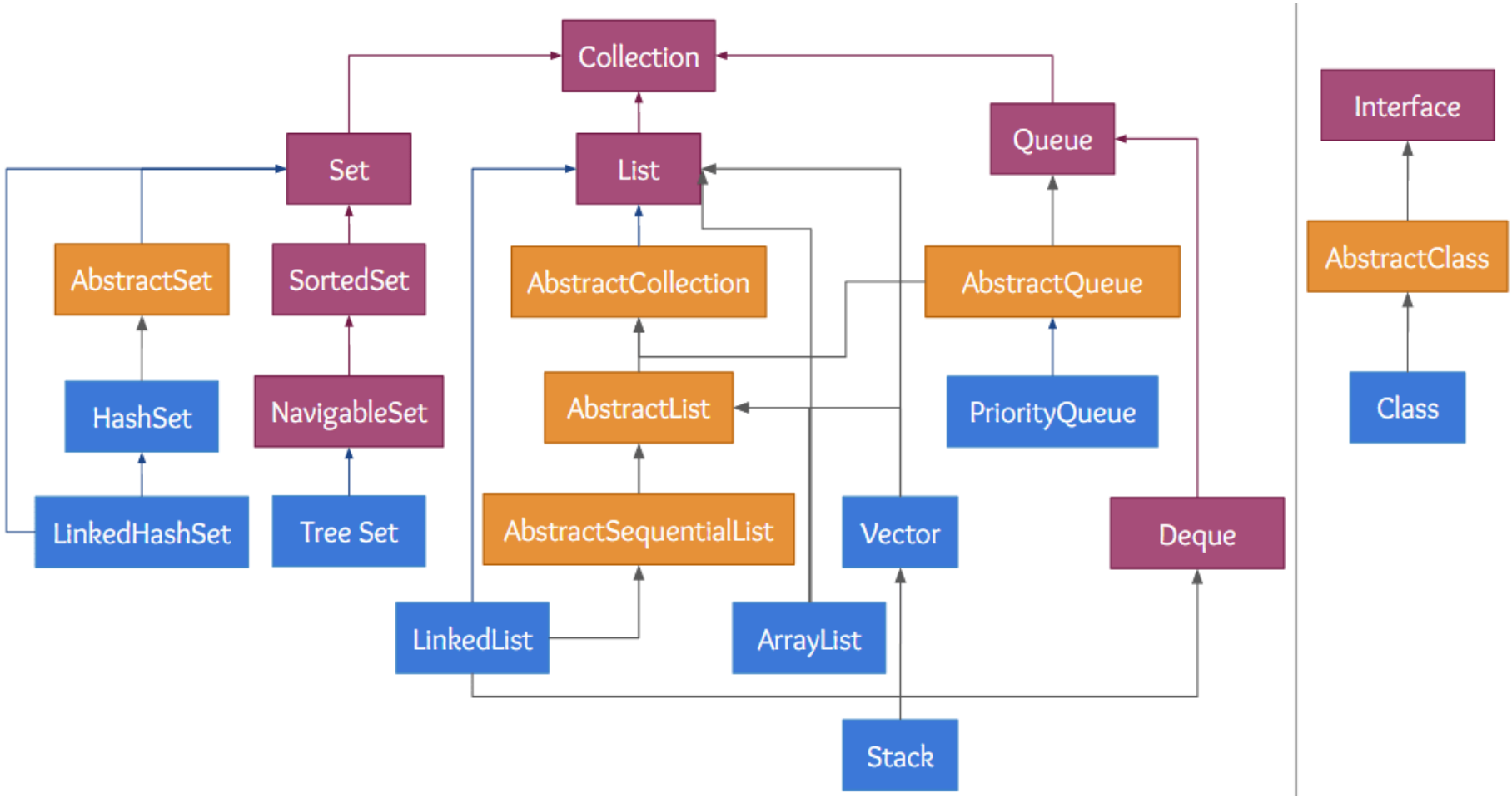
- Javadoc, provided with your java distribution.
- Comments and code in the specific java.util.*.java files
- The Collections Java tutorial, available at http://docs.oracle.com/javase/tutorial/collections/index.html
- Chan et al, The Java Class Libraries, Second Edition

# Java Collections Framework - 3

Java supports three levels of abstraction

- Interface
  - Java expression of an ADT
  - Includes method declarations with arguments of specified types, but with empty bodies
- Abstract Class
  - Implements only a subset of an interface
  - Cannot be used to instantiate an object
- Class
  - May extend one or more abstract classes
  - Must fully implement any interface it implements
  - Can be used to instantiate objects

# Java Collections Framework - 4




From: http://p3lang.com/2013/06/java-collections-framework/

# The iterable interface

- Allows an Iterator to be associated with an object.
- The iterator allows an existing data structure to be stepped through sequentially, using the following methods:
  - hasNext() returns true if the iteration has more elements
  - next() returns the next element in the iteration
    – throws exception if iterator has already visited all elements
  - remove() removes the last element that was returned by next
    - remove may be called only once per call to next otherwise throws an exception.
    - Iterator.remove is the only safe way to modify a collection during iteration

# The Collection interface

Allows data to be modeled as a collection of objects. In addition to the Iterator interface, provides interfaces for:

- Creating the data structure: `add(e)`, `addAll(c)`
- Querying the data structure: `size()`, `isEmpty()`, `contains(e)`, `containsAll(c)`, `toArray()`, `equals(c)`
- Modifying the data structure: `remove(e)`, `removeAll(c)`, `retainAll(c)`, `clear()`

# The Abstract Collection class

Allows data to be modeled as a collection of objects. In addition to the Iterator interface, provides interfaces for:

- Skeletal implementation of the Collection interface
- For unmodifiable collection, programmer still needs to implement: `iterator` (including `hasNext` and `next` methods), `size`
- For modifiable collection, need to also implement: `remove` method for iterator, `add`

# The `List` interface

Extends the Collections interface to model the data as an ordered sequence of elements, indexed by a 0-based integer index (position).

- Provides interface for creation of a ListIterator
- Several other interfaces

Creating the data structure:

- `add(e)`: append element *e* to the list
- `add(i, e)`: insert element *e* at position *i* (and shift elements at *i* and above one to the right).

# The `List` interface - 2

## Querying the data structure

- `get(i)`: return element currently in position *i*
- `indexOf(e)`: return index of first occurrence of *e*
- `lastIndexOf(e)`: return index of last occurrence of *e*
- `subList(i1, i2)`: return list of elements from *i*1 to *i*2

## Modifying the data structure

- set(i, e) – replace element currently stored at index i with specified element e
- remove(e) – remove the first occurrence of the specified element from the list
- remove(i) – remove the element at position i

# The `Abstract List` class

- Skeletal implementation of the List interface
- For unmodifiable list, programmer still needs to implement: `get`, `size`
- For modifiable collection, need to also implement: `set`
- For variable-size modifiable list, need to implement: `add`, `remove`